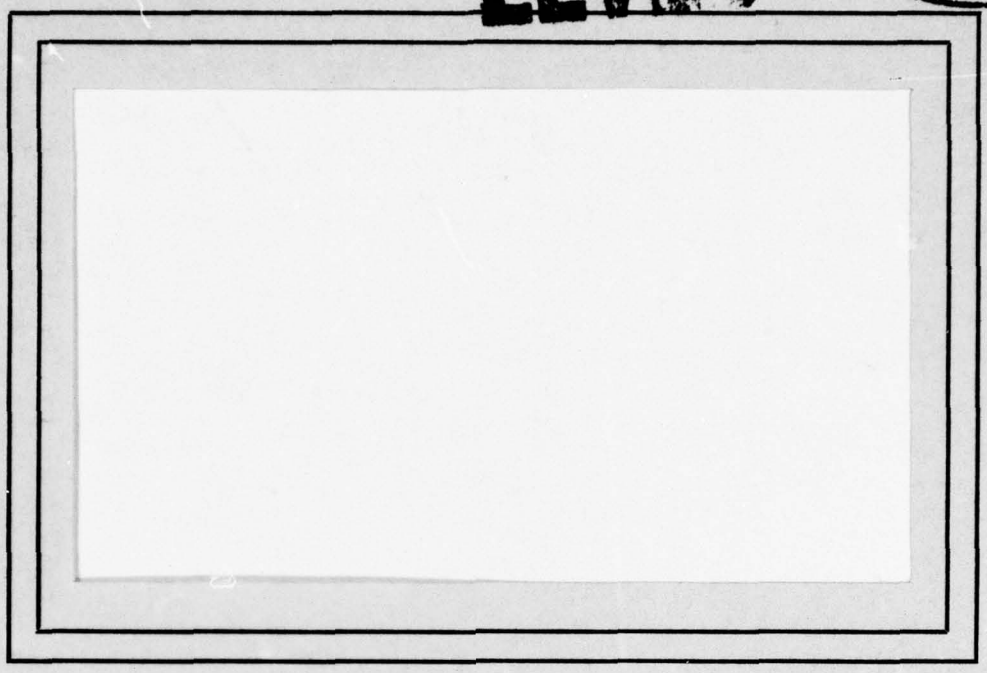


UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND

20742

(15) DAAG53-76-C-0138, DARPA Order 3206

(1)

(14) TR-755
DAAG-53-76C-0138

(11) Apr[illegible]1979

(6) COMPUTING PERIMETERS OF IMAGES REPRESENTED BY QUADTREES.

(12) 24

(10) Hanan/Samet
Computer Science Department
University of Maryland
College Park, MD 20742

(9) Technical rept.,

ABSTRACT

An algorithm is presented for computing the total perimeter of a binary image represented by a quadtree. The algorithm explores each segment of the border once and only once. Analysis of the algorithm shows that its worst-case average execution time is proportional to the product of the log of the image diameter and the number of nodes in the tree.

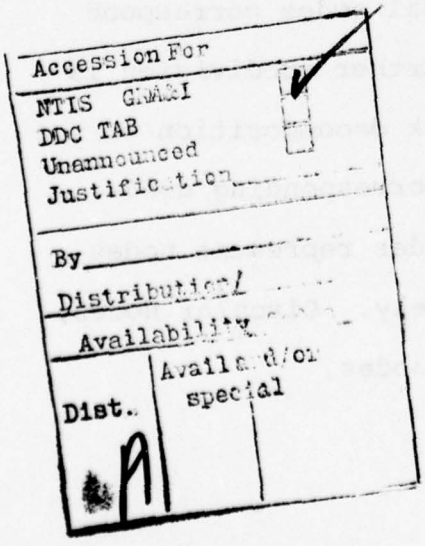
The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper. The author has benefited greatly from discussions with Charles R. Dyer and Azriel Rosenfeld. He also thanks Pat Young for her help with the figures.

1. Introduction

Perimeter computation is a basic operation in image processing [RK]. The standard algorithms use either an array or a chain code representation [Freeman] for the two-valued ("binary") image whose perimeter is to be computed. In this paper we present an algorithm for computing the total perimeter (i.e., the length of the chain codes corresponding to the black/white borders in the image) of a binary image that is represented by a quadtree ([Klinger, DRS, Samet1]).

We assume that the given image is a 2^n by 2^n array of unit square "pixels". The quadtree is an approach to image representation based on successive subdivision of the array into quadrants. In essence, we repeatedly subdivide the array into quadrants, subquadrants,... until we obtain blocks (possibly single pixels) which consist entirely of either 1's or Ø's. This process is represented by a tree of out-degree 4 in which the root node represents the entire array, the four sons of the root node represent the quadrants, and the terminal nodes correspond to those blocks of the array for which no further subdivision is necessary. For example, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree. In general, BLACK and WHITE square nodes represent nodes consisting entirely of 1's and Ø's respectively. Circular nodes, also termed GRAY nodes, denote non-terminal nodes.

Sections 2-5 present and analyze the algorithm. Included is an informal description of the algorithm along with motivating considerations. The actual algorithm is given using a variant of ALGOL 60 [Naur].



2. Definitions and Notation

Let each node in a quadtree be stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons, labeled NW, NE, SE, and SW. Given a node P and a son I, these fields are referenced as FATHER(P) and SON(P,I) respectively. At times it is useful to use the function SONTYPE(P) where SONTYPE(P) = Q iff SON(FATHER (P),Q) = P. The sixth field, named NODETYPE, describes the contents of the block of the image which the node represents--i.e., WHITE if the block contains no 1's, BLACK if the block contains only 1's, and GRAY if it contains 0's and 1's. Alternatively, BLACK and WHITE are terminal nodes while GRAY nodes are non-terminal nodes.

Let the four sides of a node's block be called its N, E, S, and W sides. They are also termed its boundaries. The spatial relationships between the various sides are specified by use of the functions OPSIDE, CSIDE, and CCSIDE. OPSIDE(B) is a side facing side B; e.g., OPSIDE(E) = W. CSIDE(B) and CCSIDE(B) correspond to the sides adjacent to side B in the clockwise and counterclockwise directions respectively; e.g., CSIDE(E) = S and CCSIDE(E) = N. We also define the following predicates and functions to facilitate the expression of operations involving a block's quadrants and boundaries. ADJ(B,I) is true if and only

if quadrant I is adjacent to boundary B of the node's block; e.g., ADJ(N,NW) is true. REFLECT(B,I) yields the quadrant which is adjacent to quadrant I along boundary B of the block represented by I; e.g., REFLECT(W,NW) = NE, REFLECT(E,NW) = NE, REFLECT(N,NW) = SW, and REFLECT(S,NW) = SW. QUAD(B,C) is the quadrant which is bounded by boundaries B and C (if B and C are not adjacent boundaries, then the value of QUAD(B,C) is undefined); e.g., QUAD(N,W) = NW. Figure 2 shows the relationship between the quadrants of a node and its boundaries.

Given a quadtree corresponding to a 2^n by 2^n array, we say that the root node is at level n, and that a node at level i is at a distance of n-i from the root of the tree. In other words, for a node at level i, we must ascend n-i FATHER links to reach the root of the tree. Note that the farthest node from the root of the tree is at level $\geq \emptyset$. A node at level $\emptyset$ corresponds to a single pixel in the image.

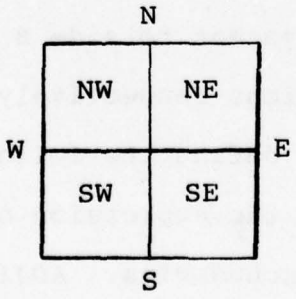


Figure 2. Relationship between a block's four quadrants and its boundaries.

3. Algorithm

The perimeter computation algorithm traverses the quadtree in postorder (i.e., the sons of a node are visited first) and visits each BLACK-WHITE border segment once and only once. For each BLACK terminal node, say P, the northern, eastern, southern, and western adjacencies are explored so that all of the node's WHITE adjacent neighbors are visited. The result of each visit is that the length of the border which is shared between the two adjacent neighbors is included in the value of the perimeter.

The main procedure is termed PERIMETER and is invoked with a pointer to the root of the quadtree representing the image and an integer corresponding to the log of the diameter of the image (e.g., n for a 2^n by 2^n image array). PERIMETER traverses the tree and controls the exploration of the adjacencies of each BLACK node. FIND_NEIGHBOR locates a neighboring node of greater or equal size along a specified side. If the node is on the edge of the image, then no neighbor exists in the specified direction and NULL is returned (e.g., border segments CO and CN in Figure 1b). If the node is not on the edge of the image and no neighboring BLACK or WHITE node exists satisfying our size criteria, then a pointer to a GRAY node of equal size is returned (e.g., the eastern border of node C in Figure 1b). In such a case, procedure SUM_ADJACENT continues the search by examining all WHITE

adjacent neighbors of smaller size and accumulating their lengths (e.g., block M for the eastern border of node C in Figure 1b). Note that when the neighboring node is BLACK and is of the same or greater size, then no contribution is made to the perimeter by the side of the node currently being examined (e.g., border segment AB in Figure 1b). A node having a side on the border of the image or having a WHITE neighboring node of the same or greater size makes a contribution to the perimeter equal to the length of the side of the BLACK node (e.g., border segment AI in Figure 1b).

An alternative method of computing the perimeter is to apply the algorithm in [DRS] which converts a quadtree representation to a chain code and simply sums the lengths of the segments. Our algorithm is simpler since it does not require the segments to be traversed in sequence around each border. We need only insure that each border segment is visited once and only once. This is clearly true since during the tree traversal, the adjacencies of each BLACK node are explored at least once; on the other hand, each border segment is only explored once since it must adjoin a WHITE node and our algorithm does not explore adjacencies of WHITE nodes.

As an example of the application of the algorithm, consider the image given in Figure 1a. Figure 1b is the corresponding block decomposition and Figure 1c is its quadtree representation.

All of the BLACK nodes have labels ranging between A and G while the WHITE nodes have labels ranging between H and S. The BLACK nodes are labeled in the order in which their adjacencies are explored by PERIMETER. WHITE nodes H through Q are labeled in the order in which they are first visited by the combination of FIND_NEIGHBOR and SUM_ADJACENT. Thus the adjacencies of node A have been explored before those of nodes B, C, etc. The value of the perimeter is obtained by visiting the border segments in the order AH, AI, BJ, BK, BL, CI, CM, CN, CO, EL, EP, FP, GQ, and GM. Assuming n=3 (i.e., blocks D, E, F, G, P, Q, R, and M are single pixels), the perimeter is 28. Note that nodes D, R, and S do not contribute to the value of the perimeter since none of their sides adjoin the border.

```
integer procedure PERIMETER(P,LEVEL);
/*find the perimeter of a quadtree rooted at node P which
  spans a 2^LEVEL by 2^LEVEL space*/
begin
    node P,Q;
    integer LEN, LEVEL;
    quadrant I;
    side S;
    LEN←Ø;
    if GRAY(P) then
        begin
            for I in {NW,NE,SW,SE} do
                LEN←LEN+PERIMETER(SON(P,I),LEVEL-1);
        end
    else if BLACK(P) then
        begin
            for S in {N,E,S,W} do
                begin
                    Q←FIND_NEIGHBOR(P,S);
                    LEN←LEN + if NULL(Q) or WHITE(Q) then 2↑LEVEL
                              else if GRAY(Q) then
                                SUM_ADJACENT(Q,QUAD(OPSIDE(S)),CSIDE(S)
                                             QUAD(OPSIDE(S),CCSIDE(S))
                                             LEVEL)
                              else Ø;
                end;
        end;
    return(LEN);
end;
```

in direction S. Of these $2^{n-i}-1$ neighbor pairs, $2^{\emptyset}$ have their nearest common ancestor at level n, 2^{1} at level n-1,... and 2^{n-i-1} at level i+1. For each node at level i having a common ancestor at level j, the maximum number of nodes that will be visited by FIND_NEIGHBOR and SUM_ADJACENT is

$$(j-i) + (j-i-1) + \sum_{k=0}^{i} 2^{k} = 2(j-i-1) + 2^{i+1}$$

This is obtained by observing that the common ancestor is at a distance of j-i and that a node at level i has a maximum of 2^{i} adjacent neighbors (all appearing at level $\emptyset$). Assuming that node P is equally likely to occur at any level i and at any of the $2^{n-i}-1$ positions at level i, then the average of the maximum number of nodes visited by FIND_NEIGHBOR and SUM_ADJACENT is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-j}(2(j-i-1)+2^{i+1})}{\sum_{i=0}^{n} (2^{n-i}-1)} \qquad (1)$$

(1) can be rewritten to yield

$$\frac{\sum_{i=0}^{n-1} \sum_{j=0}^{n-1-i} 2^{n-1-i-j}(2j+2^{i+1})}{\sum_{i=0}^{n} (2^{i}-1)} \qquad (2)$$

The numerator of (2) can be simplified as follows:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1-i} 2^{n-1-i-j}(2j+2^{i+1}) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1-i} (j\cdot 2^{n-i-j}+2^{n-j})$$

$$= \sum_{i=0}^{n-1} 2^n \sum_{j=0}^{n-1-i} \left(\frac{j}{2^{i+j}} + \frac{1}{2^j}\right) \qquad (3)$$

But $$\sum_{j=0}^{n-1-i} \frac{j}{2^{i+j}} = \frac{1}{2^i} \sum_{j=0}^{n-1-i} \frac{j}{2^j}$$

$$= \frac{1}{2^i}\left(2 - \frac{n-i}{2^{n-2-i}}\right) \qquad (4)$$

Also $$\sum_{j=0}^{n-1-i} \frac{1}{2^j} = 2\left(1 - \frac{1}{2^{n-i}}\right) \qquad (5)$$

Substituting (4) and (5) into (3) yields

$$\sum_{i=0}^{n-1} 2^n \left(\frac{1}{2^i}\left(2 - \frac{n-i}{2^{n-2-i}}\right) + 2\left(1 - \frac{1}{2^{n-i}}\right)\right)$$

$$= \sum_{i=0}^{n-1} 2^n \left(\frac{2^{n-1-i}-(n-i)}{2^{n-2}} + \frac{2^{n+1-i}-2}{2^{n-i}}\right)$$

$$= \sum_{i=0}^{n-1} (2^{n+1-i}-4(n-i)+2^{n+1}-2^{i+1})$$

$$= 2^{n+1} \sum_{i=0}^{n-1} \frac{1}{2^i} - 4 \sum_{i=0}^{n-1} (n-i) + n\cdot 2^{n+1} - \sum_{i=0}^{n-1} 2^{i+1}$$

$$= 2^{n+1}\left(2\left(1 - \frac{1}{2^n}\right)\right) - 4\frac{n(n+1)}{2} + n\cdot 2^{n+1} - 2^{n+1} + 2$$

$$= 2^{n+2} - 4 - 2n^2 - 2n + (n-1)2^{n+1} + 2$$

$$= 2^{n+1}(n+1) - 2(n^2+n+1) \qquad (6)$$

4. Analysis

The running time of the perimeter computation algorithm, measured by the number of nodes visited, depends on the time spent locating adjacent WHITE nodes and on the size of the quadtree. Adjacent WHITE nodes are located by procedures FIND_NEIGHBOR and SUM_ADJACENT. They are invoked once for each BLACK node. The amount of work performed by these procedures is obtained by considering the number of nodes that are visited when an adjacency is being explored. Recall that we must find the neighbor, and if it is GRAY, then visit all adjacent WHITE neighbors of smaller size. In the worst case we are at level n-1, with a GRAY neighbor, and all adjacent neighbors are at level Ø. In such a case, we must visit 2^n nodes. For example, consider Figure 1b where n=3 and we wish to visit the blocks adjacent to the block labeled C (i.e., blocks D, F, G, and M). We must visit the root of the quadtree as well as A's neighboring GRAY node and all of its NW and SW sons--i.e., a complete binary tree of height 2. In total, 2^3=8 nodes are visited. Assuming a 2^n by 2^n random image--i.e., a BLACK node is equally likely to appear in any position and level in a quadtree, we have the following result.

Theorem 1: The average of the maximum number of nodes visited by each invocation of FIND_NEIGHBOR and SUM_ADJACENT is n+1.

Proof: Given a node P at level i and a direction S, there are $2^{n-i}-1$ possible positions for node P and a neighbor at level i

```
node procedure FIND_NEIGHBOR(P,S);
/*given node P, return a node which is adjacent to side S of node P*/
begin
    node P,Q;
    side S;
    if not NULL(FATHER(P)) and ADJ(S,SONTYPE(P)) then
        /* find a common ancestor* /
        Q←FIND_NEIGHBOR(FATHER(P),S)
    else Q←FATHER(P);
    /*follow reflected path to locate the neighbor*/
    return (if not NULL(Q) and GRAY(Q) then SON(Q,REFLECT(S,SONTYPE(P)))
            else Q);
end;

integer procedure SUM_ADJACENT(P,Q1,Q2,LEVEL);
/*find all WHITE descendants of node P adjacent to the perimeter--
  i.e., in quadrants Q1 and Q2, and return the length of their s
begin
    node P;
    quadrant Q1,Q2
    integer LEVEL;
    return (if GRAY(P) then SUM_ADJACENT(SON(P,Q1),Q1,Q2,LEVEL-1)
                            SUM_ADJACENT(SON(P,Q2),Q1,Q2,LEVEL-1
            else if WHITE(P) then 2↑LEVEL
            else Ø);
end;
```

The denominator of (2) can be simplified as follows:

$$\sum_{i=0}^{n} (2^i-1) = 2^{n+1}-1-(n+1)$$

or

$$\sum_{i=0}^{n} (2^i-1) = 2^{n+1}-n-2 \qquad (7)$$

Substituting (6) and (7) into (2) yields

$$\frac{2^{n+1}(n+1)-2(n^2+n+1)}{2^{n+1}-n-2} = n+1 - \frac{n^2-n}{2^{n+1}-n-2}$$

$$\approx n+1 \quad \text{as } n \text{ gets large}$$

Q.E.D.

It is useful to obtain an upper bound on the size of the quadtree in terms of the number of BLACK nodes. Letting B denote the number of BLACK nodes we have

Lemma 1: The maximum number of nodes in a quadtree having B black nodes is 4Bn+1.

Proof: Given B BLACK nodes at level Ø there is a maximum of 3B WHITE nodes at level Ø. At worst there is one GRAY node at level 1 for every BLACK node and three WHITE nodes at level Ø, and at worst three additional WHITE nodes for each such GRAY node. Thus there exist at most 4B nodes at level 1. Repeating the same argument for levels 2 through n-1, we have 4Bn nodes. At level n there is only one node. Therefore, the maximum number of nodes is 4Bn+1.

Q.E.D.

We can now prove

Theorem 2: The average worst case execution time of the perimeter computation algorithm has an upper bound proportional to the product of the number of BLACK nodes and the log of the diameter of the image.

Proof: From Theorem 1 we have that for each adjacency involving a BLACK node, FIND_NEIGHBOR and SUM_ADJACENT result in an average worst case of n+1 nodes being visited. There are four adjacencies for each BLACK node. Thus these two procedures contribute $4B(n+1)$. From Lemma 1 we have that the number of nodes in the quadtree is bounded by $4Bn+1$. However, this quantity correlates with the work performed by procedure PERIMETER since each node in the quadtree is visited by the traversal. Summing up these values we have $4B(n+1) + 4Bn+1 = 8Bn+4B+1$.

Q.E.D.

6. References

[DRS] C. R. Dyer, A. Rosenfeld, and H. Samet. Region representation: boundary codes from quadtrees. Computer Science TR-732, University of Maryland, College Park, Maryland, February 1979.

[Freeman] H. Freeman, Computer processing of line-drawing images, Computing Surveys 6, 1974, 57-97.

[Klinger] A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

[Naur] P. Naur (Ed.), Revised report on the algorithmic language ALGOL 60, Communications of the ACM 3, 1960, 299-314.

[RK] A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976, Section 9.2.1.

[Samet1] H. Samet, Region representation: quadtrees from boundary codes, Computer Science TR-741, University of Maryland, College Park, Maryland, March 1979.

[Samet2] H. Samet, Connected component labeling using quadtrees, Computer Science TR-756, University of Maryland, College Park, Maryland, April 1979.

5. Concluding remarks

An algorithm has been presented for computing the total perimeter of a binary image represented by a quadtree. The algorithm's execution time has been shown to have an average worst case time complexity proportional to the product of the image's diameter and the number of BLACK nodes in the quadtree representation of the image. It should be clear that if the image has more than one connected component, the algorithm returns the total perimeter of all the regions. Similarly, if holes are present, their boundaries are also included in the value of the perimeter obtained by this algorithm. Note that if we first labeled the connected components of the image [Samet2], then the perimeter of each boundary could be separately computed.

The algorithm demonstrates the utility of the quadtree as a desirable data structure for image representation. Computation of perimeter is generally achieved by use of a chain code representation. We have shown that it can be computed with reasonable efficiency when the quadtree is used as the data structure.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) COMPUTING PATTERNS OF IMAGES REPRESENTED BY QUADTREES		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-755
7. AUTHOR(s) Hanan Samet		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Night Vision Laboratory Fort Belvoir, VA 22060		12. REPORT DATE April 1979
		13. NUMBER OF PAGES 18
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Region representation
Quadtrees
Perimeter

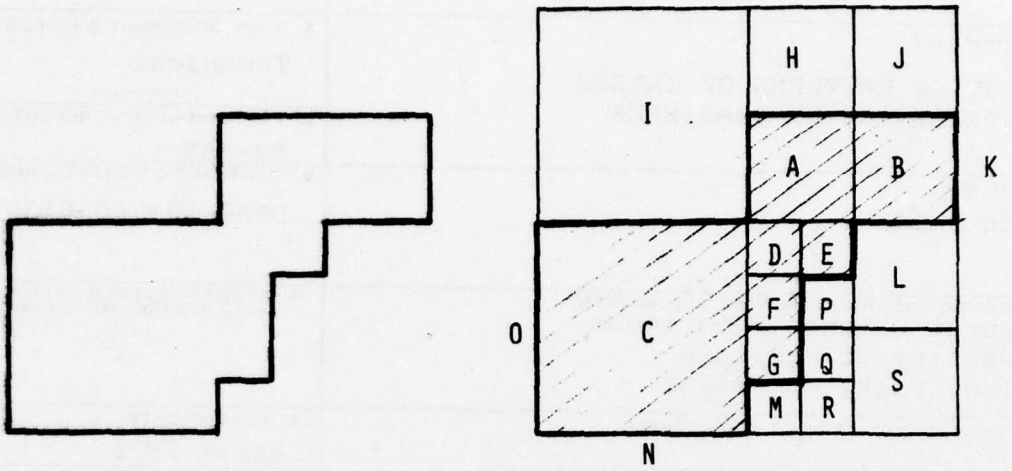
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm is presented for computing the total perimeter of a binary image represented by a quadtree. The algorithm explores each segment of the border once and only once. Analysis of the algorithm shows that its worst-case average execution time is proportional to the product of the log of the image diameter and the number of nodes in the tree.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

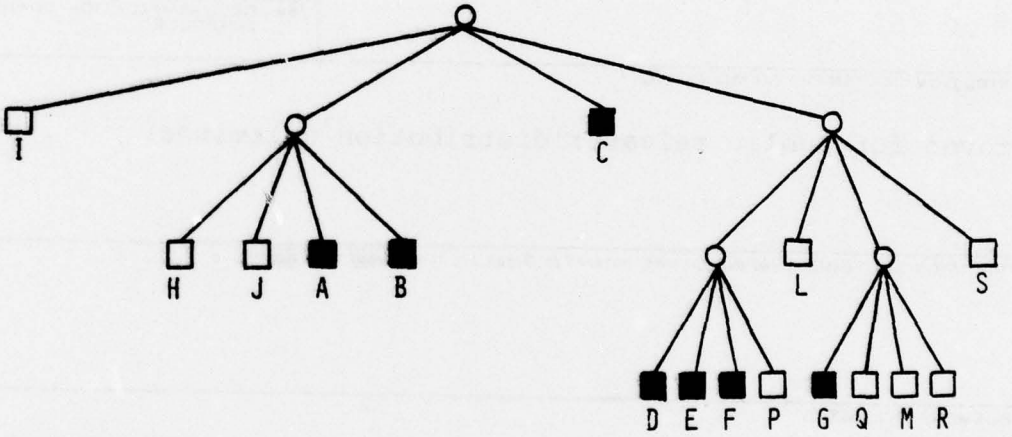
Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



a. SAMPLE IMAGE

b. BLOCK DECOMPOSITION OF THE IMAGE IN (a).



c. QUADTREE REPRESENTATION OF THE BLOCKS IN (b).

FIG. 1. AN IMAGE, ITS MAXIMAL BLOCKS, AND THE CORRESPONDING QUADTREE. BLOCKS IN THE IMAGE ARE SHADED.